LPI exam 101 prep: Hardware and architecture

Junior Level Administration (LPIC-1) topic 101

Skill Level: Introductory

Ian Shields (ishields@us.ibm.com)
Senior Programmer
IBM

08 Aug 2005

In this tutorial, Ian Shields begins preparing you to take the Linux Professional Institute® Junior Level Administration (LPIC-1) Exam 101. In this first of five tutorials, Ian introduces you to configuring your system hardware with Linux™. By the end of this tutorial, you will know how Linux configures the hardware found on a modern PC and where to look if you have problems.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at two levels: *junior level* (also called "certification level 1") and *intermediate level* (also called "certification level 2"). To attain certification level 1, you must pass exams 101 and 102; to attain certification level 2, you must pass exams 201 and 202.

developerWorks offers tutorials to help you prepare for each of the four exams. Each exam covers several topics, and each topic has a corresponding self-study tutorial on developerWorks. For LPI exam 101, the five topics and corresponding developerWorks tutorials are:

Table 1. LPI exam 101: Tutorials and topics

LPI exam 101 topic	developerWorks tutorial	Tutorial summary
Topic 101	LPI exam 101 prep (topic	(This tutorial). Learn to

	101): Hardware and architecture	configure your system hardware with Linux. By the end of this tutorial, you will know how Linux configures the hardware found on a modern PC and where to look if you have problems.
Topic 102	LPI exam 101 prep: Linux installation and package management	Get an introduction to Linux installation and package management. By the end of this tutorial, you will know how Linux uses disk partitions, how Linux boots, and how to install and manage software packages.
Topic 103	LPI exam 101 prep: GNU and UNIX commands	Get an introduction to common GNU and UNIX commands. By the end of this tutorial, you will know how to use commands in the bash shell, including how to use text processing commands and filters, how to search files and directories, and how to manage processes.
Topic 104	LPI exam 104 prep: Devices, Linux filesystems, and the Filesystem Hierarchy Standard.	Learn how to create filesystems on disk partitions, as well as how to make them accessible to users, manage file ownership and user quotas, and repair filesystems as needed. Also learn about hard and symbolic links, and how to locate files in your filesystem and where files should be placed. See detailed objectives below.
Topic 110	The X Window system	Coming soon.

To pass exams 101 and 102 (and attain certification level 1), you should be able to:

- Work at the Linux command line
- Perform easy maintenance tasks: help out users, add users to a larger system, back up and restore, and shut down and reboot
- Install and configure a workstation (including X) and connect it to a LAN, or connect a stand-alone PC via modem to the Internet

To continue preparing for certification level 1, see the developerWorks tutorials for LPI exam 101. Read more about the entire set of developerWorks LPI tutorials.

The Linux Professional Institute does not endorse any third-party exam preparation

material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "Hardware and architecture," the first of five tutorials designed to prepare you for LPI exam 101. In this tutorial, you will learn about PC hardware and architecture.

This tutorial is organized according to the LPI objectives for this topic. Very roughly, expect more questions on the exam for objectives with higher weight.

Table 2. Hardware and architecture: Exam objectives covered in this tutorial

LPI exam objective	Objective weight	Objective summary
1.101.1 Configure fundamental BIOS settings	Weight 1	You will learn to configure fundamental system hardware by making the correct settings in the system BIOS. You will learn about configuration issues such as the use of LBA on IDE hard disks larger than 1024 cylinders, enabling or disabling integrated peripherals, and configuring systems with (or without) external peripherals such as keyboards. We also discuss correct settings for IRQ, DMA, and I/O addresses for all BIOS-administered ports and settings for error handling.
1.101.3 Configure modem and sound cards	Weight 1	You will learn how to ensure that devices meet compatibility requirements and how to set up both the modem and sound card. You will learn how to configure a modem for outbound dialup, and how to use it for outbound PPP, SLIP, or CSLIP connections.
1.101.4 Set up SCSI devices	Weight 1)	You will learn how to configure SCSI devices using the SCSI BIOS as well as the necessary Linux tools. You will review the various types of SCSI. You will learn how to set up a SCSI boot device and how to set the desired boot sequence in a mixed SCSI and IDE environment.
1.101.5 Set up different PC expansion cards	Weight 3)	You will learn about the differences between ISA and PCI cards with respect to configuration issues. You will

		learn how to check the settings of IRQs, DMAs, and I/O ports to avoid conflicts between devices.
1.101.6 Configure communication devices	Weight 1	You will learn how to install and configure different internal and external communication devices such as modems, ISDN adapters, and DSL switches. You will learn about compatibility requirements (especially important if that modem is a winmodem), necessary hardware settings for internal devices (IRQs, DMAs, I/O ports), and loading and configuring suitable device drivers. We will also cover interface configuration requirements.
1.101.7 Configure USB devices	Weight 1	You will learn how to activate USB support and how to use and configure different USB devices. You will learn about correct selection of your USB chipset and the corresponding module. We will also cover the basic architecture of the layer model of USB and the different modules used in the different layers.

Prerequisites

There are no formal prerequisites for this tutorial. To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

Different versions of a program may format output differently, so your results may not look exactly like the listings and figures in this tutorial.

Section 2. BIOS settings

This section covers material for topic 1.101.1 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

We will start with a high-level overview of a modern personal computer, and then we'll discuss the configuration issues for setting up a system. We will focus on

systems using an x86 processor, such as an Intel® Pentium® or AMD Athlon processor, and a PCI bus, as these are the most common today.

Many of the topics covered here have a high level of overlap with LPI objectives for specific peripherals. Later sections of this tutorial will refer you back to this section for basic material.

System and BIOS overview

A modern personal computer (or PC) system consists of a central processing unit (CPU) for performing calculations, along with some memory for storing the data that the processor is using. To make such a device useful, we attach peripheral devices, such as keyboards, mice, displays, hard drives, CD or DVD drives, printers, scanners, and network cards, which allow us to enter, store, print, display, and transmit data.

In the computer just described, the memory used by the processor is called Random Access Memory (RAM). In a typical PC, this memory is *volatile*, meaning that it requires power to keep its data. Turn off the PC and the memory is wiped clean. Put another way, when we turn off a PC, we turn it into a collection of hardware components that will do nothing until reprogrammed. This reprogramming occurs when we turn on the machine; the process is called *bootstrapping* or *booting* the computer.

Bootstrap process and BIOS

The process of booting involves loading an operating system from an external storage device, such as a floppy disk, CD, DVD, hard drive, or memory key. The program that does this initial loading is permanently stored in the computer and is called the *Basic Input Output System (BIOS)*. The BIOS is stored in non-volatile memory, sometimes called *Read Only Memory (ROM)*. In early PCs, the ROM chip was often soldered or socketed to the computer main board (or *motherboard*). Updating the BIOS meant replacing the ROM chip. Later, *Electrically Erasable Programmable Read Only Memories (EEPROMs)* were used. EEPROMs allowed BIOS to be upgraded in the field with a diskette instead of special tools. Today you will more often find a form of non-volatile memory known as *Flash* memory, which is also used in digital cameras and memory keys. Flash memory also permits BIOS upgrades in the field.

Besides controlling the initial bootup of a PC, today's BIOS programs usually permit a user to set or verify several configuration options on a system. These include verifying installed features such as RAM, hard drive, optical drive, keyboard, mouse, and possibly onboard display, sound and network connections. The user may enable or disable some features. For example, the onboard sound may be disabled to allow use of an installed sound card. The user may also choose which devices will be considered for booting the system and whether the system is protected by a password.

Accessing the BIOS setup screens usually requires a keyboard to be attached to the

system. When a system is powered on a *Power On Self Test* or *POST* is performed. On some systems you will be briefly prompted to press a particular key to enter setup otherwise normal bootup takes over. On other systems you will need to know which key to press before the normal boot process is invoked as the prompt is either not present or may have been removed as the result of previous customization of setup options. On some systems you may have other choices besides going to the BIOS setup, such as illustrated in Figure 1. Otherwise, you should see a BIOS summary screen such as that shown in Figure 2.

Figure 1. Accessing the BIOS settings

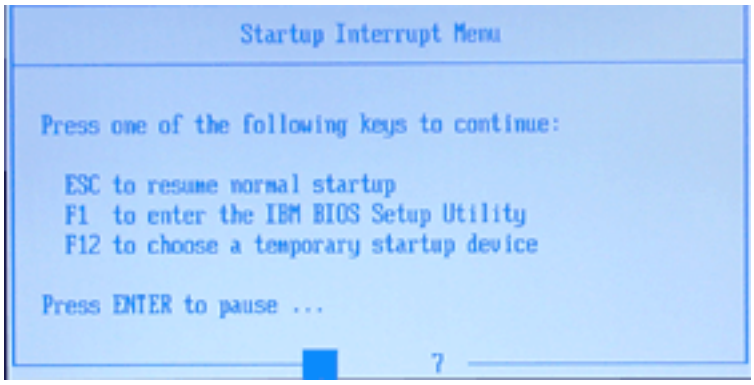
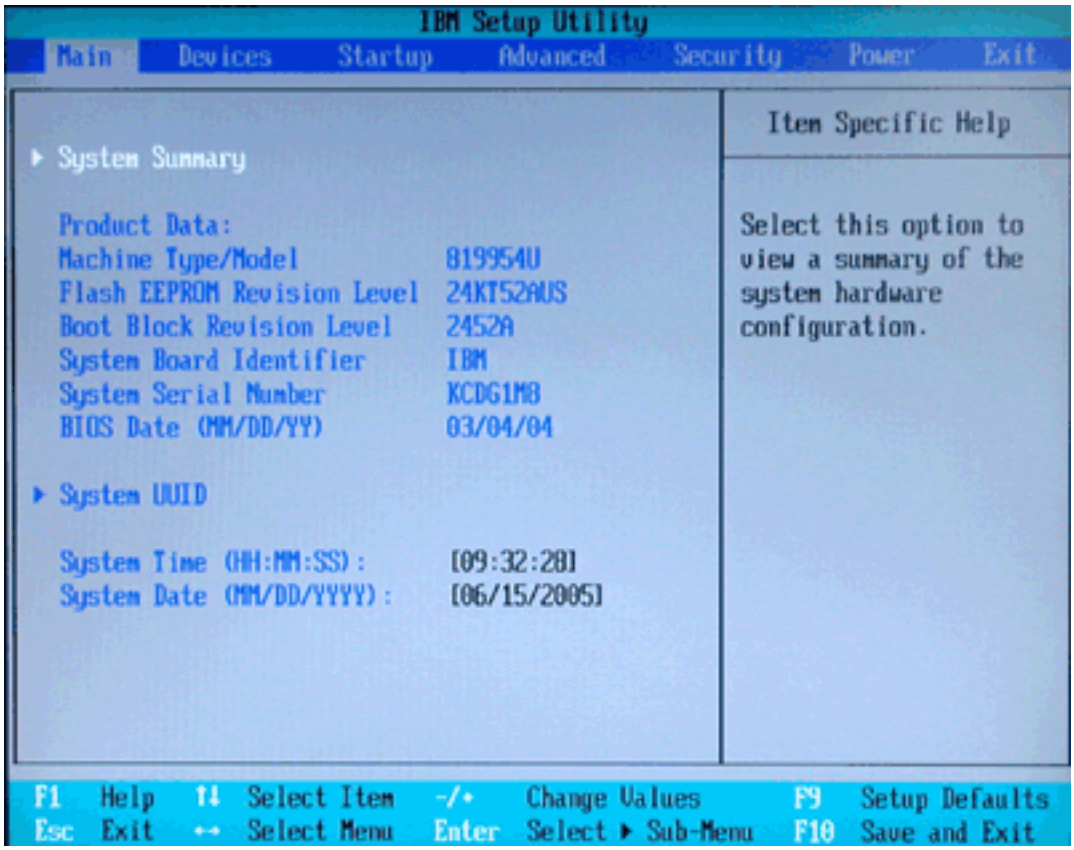


Figure 2. BIOS settings summary



The above illustrations are examples of what you may see, but BIOS setup screens vary widely, so don't be surprised if yours looks different.

Figure 2 shows us that the Flash EEPROM (or system BIOS) revision level is

24KT52AUS and it is dated March 4, 2004 while the current date on the system is June 9, 2005. A check on the manufacturer's (IBM) support site shows that several later BIOS versions are available, so it would probably be a good idea to upgrade this system's BIOS.

You will notice several other menu selections in Figure 2. We will cover these in the remaining sections of this tutorial. Before we do though, let's review a little more of the inner workings of a PC.

Buses, ports, IRQs, and DMA.

PCI and ISA buses

Peripheral devices, including those that may be built in to the system board, communicate with the CPU over a *bus*. The most common bus type in use today is the *Peripheral Component Interconnect* or *PCI* bus which has mostly superseded the earlier *Industry Standard Architecture* or *ISA* bus. The ISA bus was sometimes called the *AT* bus after the IBM PC-AT in which it was first used in 1984. During the transition from ISA to PCI bus, many systems included both buses with slots permitting the use of either ISA or PCI peripherals.

The ISA bus supports 8-bit and 16-bit cards, while the PCI bus support 32-bit devices.
There are a couple of other bus standards that you should also know about. Many systems include an *Accelerated Graphics Port* or *AGP* slot which is a special slot based on the PCI 2.1 bus specification, but optimized for the high bandwidth and fast response required for graphics cards. This is slowly being replaced by the newer *PCI Express* or *PCI-E* bus which addresses many limitations of the original PCI design.

We'll learn more about the Linux file system in later tutorials in this series, but right now we'll introduce you the */proc* filesystem. This is not a real filesystem on disk, but a "pseudo file system" which provides information about the running system. Within this file system, the file */proc/pci* contains information about the devices on the system's PCI bus. There has been some discussion about discontinuing this particular file, as the `lspci` command gives similar information. Run the command `cat /proc/pci` to see output which will look something like Listing 1.

Listing 1. /proc/pci

```
PCI devices found:
  Bus  0, device   0, function  0:
    Host bridge: Intel Corp. 82845G/GL [Brookdale-G] Chipset Host Bridge
         (rev 1).
      Prefetchable 32 bit memory at 0xd0000000 [0xdfffffff].
  Bus  0, device   2, function  0:
    VGA compatible controller: Intel Corp. 82845G/GL [Brookdale-G] Chipset
         Integrated Graphics Device (rev 1).
      IRQ 11.
      Prefetchable 32 bit memory at 0x88000000 [0x8fffffff].
      Non-prefetchable 32 bit memory at 0x80000000 [0x8007ffff].
  Bus  0, device  29, function  0:
```

```
    USB Controller: Intel Corp. 82801DB USB (Hub #1) (rev 1).
      IRQ 11.
      I/O at 0x1800 [0x181f].
  Bus  0, device  29, function  1:
    USB Controller: Intel Corp. 82801DB USB (Hub #2) (rev 1).
      IRQ 10.
      I/O at 0x1820 [0x183f].
  Bus  0, device  29, function  2:
    USB Controller: Intel Corp. 82801DB USB (Hub #3) (rev 1).
      IRQ 5.
      I/O at 0x1840 [0x185f].
  Bus  0, device  29, function  7:
    USB Controller: Intel Corp. 82801DB USB2 (rev 1).
      IRQ 9.
      Non-prefetchable 32 bit memory at 0xc0080000 [0xc00803ff].
  Bus  0, device  30, function  0:
    PCI bridge: Intel Corp. 82801BA/CA/DB/EB PCI Bridge (rev 129).
      Master Capable.  No bursts.  Min Gnt=4.
  Bus  0, device  31, function  0:
    ISA bridge: Intel Corp. 82801DB LPC Interface Controller (rev 1).
  Bus  0, device  31, function  1:
    IDE interface: Intel Corp. 82801DB Ultra ATA Storage Controller
         (rev 1).
      IRQ 5.
      I/O at 0x1860 [0x186f].
      Non-prefetchable 32 bit memory at 0x60000000 [0x600003ff].
  Bus  0, device  31, function  3:
    SMBus: Intel Corp. 82801DB/DBM SMBus Controller (rev 1).
      IRQ 9.
      I/O at 0x1880 [0x189f].
  Bus  0, device  31, function  5:
    Multimedia audio controller: Intel Corp. 82801DB AC'97 Audio
         Controller (rev 1).
      IRQ 9.
      I/O at 0x1c00 [0x1cff].
      I/O at 0x18c0 [0x18ff].
      Non-prefetchable 32 bit memory at 0xc0080c00 [0xc0080dff].
      Non-prefetchable 32 bit memory at 0xc0080800 [0xc00808ff].
  Bus  2, device   8, function  0:
    Ethernet controller: Intel Corp. 82801BD PRO/100 VE (LOM) Ethernet
         Controller (rev 129).
      IRQ 9.
      Master Capable.  Latency=66.  Min Gnt=8.Max Lat=56.
      Non-prefetchable 32 bit memory at 0xc0100000 [0xc0100fff].
      I/O at 0x2000 [0x203f].
```

You might want to compare this with the output from the `lspci` command. This is usually on the path of the root user, but non-root users will probably need to give the full path `/sbin/lspci`. Try these on your own system.

IO Ports

When the CPU needs to communicate with a peripheral device it does so through an *IO port* or sometimes just simply *port*.When the CPU wants to send data or control information to the peripheral, it writes to a port. When the device has data or status ready for the CPU, the CPU reads the data or status from a port. Most devices have more than one port associated with them, typically a small power of 2, such as 8, 16 or 32. Data transfer is usually done a byte or two at a time. Devices cannot share ports, so if you have ISA cards, you must ensure that each device has its own port or ports assigned. Originally, this was done using switches or jumpers on the card. Some later ISA cards used a system called *Plug and Play* or *PnP* which will discuss later in this section. PCI cards all have PnP configuration.

Within the /proc file system, the file /proc/ioports tells us about the IO ports available

on the system. Run the command `cat /proc/ioports` to see output which will look something like Listing 2.

Listing 2. /proc/ioports

```
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)
0376-0376 : ide1
0378-037a : parport0
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
1800-181f : Intel Corp. 82801DB USB (Hub #1)
  1800-181f : usb-uhci
1820-183f : Intel Corp. 82801DB USB (Hub #2)
  1820-183f : usb-uhci
1840-185f : Intel Corp. 82801DB USB (Hub #3)
  1840-185f : usb-uhci
1860-186f : Intel Corp. 82801DB Ultra ATA Storage Controller
  1860-1867 : ide0
  1868-186f : ide1
1880-189f : Intel Corp. 82801DB/DBM SMBus Controller
18c0-18ff : Intel Corp. 82801DB AC'97 Audio Controller
  18c0-18ff : Intel ICH4
1c00-1cff : Intel Corp. 82801DB AC'97 Audio Controller
  1c00-1cff : Intel ICH4
2000-203f : Intel Corp. 82801BD PRO/100 VE (LOM) Ethernet Controller
  2000-203f : e100
```

The port numbers are in hexadecimal (base 16). You'll doubtless see several that look familiar, such as keyboard, timer, parallel (printer), serial (modem) and display (vga+). Compare these with the some of the standard IO port assignments for a PC as shown in Listing 3. Notice, for example, that the first parallel port is (parport0) has the address range 0378 to 037A allocated in the /proc/ioports listing, but the standard allows it (LPT!) to use the range 378 through 37F.

Listing 3. Standard I/O Port Settings

```
1FO-1F8 - Hard Drive Controller, 16-bit ISA
200-20F - Game Control
210 - Game I/O
220 - Soundcard
278-27F - LPT2
2F8-2FF - COM2
320-32F - Hard Drive Controller, 8-bit ISA
378-37F - LPT1
3B0-3BF - Monochrome Graphics Adapter (MGA)
3D0-3DF - Colour Graphics Adapter (CGA)
3F0-3F7 - Floppy Controller
3F8-3FF - COM1
```

Interrupts

So how does the CPU know when the last output is finished or when data is waiting to be read? Usually, this information is available in a status register which may be accessed by reading one (or more) of the IO ports associated with a device. Two obvious problems arise with this scenario. Firstly, the CPU has to spend time checking the status. Secondly, if the device has data coming from somewhere, such as an attached modem, the data must be read by the CPU in a timely fashion otherwise it might be overwritten by the next available data byte.

The dual problems of not wasting unnecessary CPU cycles and ensuring that data is read or written in a timely fashion are addressed by the concept of *interrupts*. Interrupts are also called *Interrupt Requests* or *IRQs*. When something happens in a device that the CPU needs to know about, the device raises an interrupt and the CPU temporarily stops whatever else it was doing to deal with the situation.

With our experience from the last section, it should hardly come as a surprise that information on interrupts is also kept in the /proc file system, in /proc/interrupts. Run the command `cat /proc/interrupts` to see output which will look something like Listing 4.

Listing 4. /proc/interrupts

```
           CPU0
  0:  226300426          XT-PIC  timer
  1:      92913          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  5:          0          XT-PIC  usb-uhci
  8:          1          XT-PIC  rtc
  9:    2641134          XT-PIC  ehci-hcd, eth0, Intel ICH4
 10:          0          XT-PIC  usb-uhci
 11:     213632          XT-PIC  usb-uhci
 14:    1944208          XT-PIC  ide0
 15:    3562845          XT-PIC  ide1
NMI:          0
ERR:          0
```

This time, the interrupt numbers are decimal in the range 0 through 15. Once again, Compare these with the standard IRQ assignments for a PC as shown in Listing 5.

Listing 5. Standard IRQ Settings

```
IRQ 0 - System Timer
IRQ 1 - Keyboard
IRQ 2(9) - Video Card
IRQ 3 - COM2, COM4
IRQ 4 - COM1, COM3
IRQ 5 - Available (LPT2 or Sound Card)
IRQ 6 - Floppy Disk Controller
IRQ 7 - LPT1
IRQ 8 - Real-Time Clock
IRQ 9 - Redirected IRQ 2
IRQ 10 - Available
IRQ 11 - Available
IRQ 12 - PS/2 Mouse
IRQ 13 - Math Co-Processor
IRQ 14 - Hard Disk Controller
IRQ 15 - Available
```

Originally, each device had its own private IRQ. In Listing 5, note, for example, that

IRQ5 was often used for **either** a sound card or a second parallel (printer) port. If you wanted both, you had to find a card that could be configured (usually via hardware jumper settings) to use another IRQ such as IRQ15.

Today, PCI devices share IRQs, so that when one interrupts the CPU, an interrupt handler checks to see if the interrupt is for it and, if not, passes it to the next handler in the chain. Listings 4 and 5 do not tell us about this sharing. We will learn about the `grep` command in a later tutorial, but for now we can use it to filter the output from the `dmesg` command to look for bootstrap messages about IRQs as shown in Listing 6. we've highlighted the shared interrupts here.

Listing 6. Interrupts found during bootstrap

```
[ian@lyrebird ian]$ dmesg | grep -i irq
PCI: Discovered primary peer bus 01 [IRQ]
PCI: Using IRQ router PIIX [8086/24c0] at 00:1f.0
PCI: Found IRQ 5 for device 00:1f.1
PCI: Sharing IRQ 5 with 00:1d.2
Serial driver version 5.05c (2001-07-08) with MANY_PORTS MULTIPORT
     SHARE_IRQ SERIAL_PCI ISAPNP enabled
ttyS0 at 0x03f8 (irq = 4) is a 16550A
ttyS1 at 0x02f8 (irq = 3) is a 16550A
PCI: Found IRQ 5 for device 00:1f.1
PCI: Sharing IRQ 5 with 00:1d.2
ICH4: not 100% native mode: will probe irqs later
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
ide1 at 0x170-0x177,0x376 on irq 15
PCI: Found IRQ 11 for device 00:1d.0
PCI: Sharing IRQ 11 with 00:02.0
usb-uhci.c: USB UHCI at I/O 0x1800, IRQ 11
PCI: Found IRQ 10 for device 00:1d.1
usb-uhci.c: USB UHCI at I/O 0x1820, IRQ 10
PCI: Found IRQ 5 for device 00:1d.2
PCI: Sharing IRQ 5 with 00:1f.1
usb-uhci.c: USB UHCI at I/O 0x1840, IRQ 5
PCI: Found IRQ 9 for device 00:1d.7
ehci-hcd 00:1d.7: irq 9, pci mem f885d000
parport0: irq 7 detected
PCI: Found IRQ 9 for device 02:08.0
PCI: Found IRQ 9 for device 02:08.0
parport0: irq 7 detected
PCI: Found IRQ 11 for device 00:02.0
PCI: Sharing IRQ 11 with 00:1d.0
PCI: Found IRQ 9 for device 00:1f.5
PCI: Sharing IRQ 9 with 00:1f.3
i810: Intel ICH4 found at IO 0x18c0 and 0x1c00, MEM 0xc0080c00 and
     0xc0080800, IRQ 9
```

DMA

We mentioned earlier that communication with peripheral devices through IO ports occurs a byte or two at a time. For a fast device, servicing interrupts could use a lot of the CPUs capability. A faster method is to use *Direct Memory Access* or *DMA*, in which a few IO instructions tell the device where in RAM to read or write data and then the DMA controller provides hardware management of the actual transfer of data between RAM and the peripheral device.

Hands up anyone who can guess where we find information about the DMA channels are in use. If you said it is in /proc/dma, then you are right. Run the command `cat /proc/dma` to see output which will look something like Listing 7.

Listing 7. /proc/dma

```
 4: cascade
```

Is that all? It is important to remember that most devices will only request one of the limited number of DMA channels when IO is actually happening, so /proc/dma will frequently look nearly empty as in our example. We can also scan the bootstrap messages for evidence of DMA capable devices as we did for IRQs above. Listing 8 shows typical output.

Listing 8. /proc/dma

```
[ian@lyrebird ian]$ dmesg | grep -i dma
    ide0: BM-DMA at 0x1860-0x1867, BIOS settings: hda:DMA, hdb:pio
    ide1: BM-DMA at 0x1868-0x186f, BIOS settings: hdc:DMA, hdd:DMA
hda: 312581808 sectors (160042 MB) w/8192KiB Cache,
     CHS=19457/255/63, UDMA(100)
hdc: 398297088 sectors (203928 MB) w/7936KiB Cache,
     CHS=24792/255/63, UDMA(33)
ehci-hcd 00:1d.7: enabled 64bit PCI DMA
```

Plug and play

Early PCs allocated fixed port numbers and IRQs for particular devices, such as keyboard or parallel printer port. This made it difficult to add new devices or even run two devices of the same type such as two modems or two printers. The first serial port was usually called COM1 and the second COM2. Linux systems usually refer to these as *ttyS0* and *ttyS1*. Some cards were configurable usually with hardware jumpers which allowed a modem to operate as either COM1 or COM2, for example. As devices proliferated and the original space allocated for IO port addresses and IRQs became scarce, *Plug and Play* or *PnP* was developed. The idea was to allow a device to tell the system how many and what kind of resources it needed and for the BIOS to then tell the device which particular resources it should use. This semi-automatic configuration was introduced with the IBM PS/2 which used a bus architecture called *microchannel*. Later, the idea, and the plug and play name were used for ISA cards, particularly modems and sound cards which were popular add-on cards at the time. The PCI bus advanced the idea further and all PCI devices are inherently plug and play.

If you happen to work on a system with ISA PnP devices, be aware that you must avoid port and IRQ conflicts between devices. Ports cannot be shared between two devices; each device **must** have its own ports. The same applies for DMA channels. With few exceptions, ISA devices cannot share IRQs either. If you have non-PnP devices, you must manually configure each device so that it does not interfere with another device. The promise of PnP was that configuration could be performed automatically. However, with some ISA devices not participating in PnP, this does not always work perfectly. You may be able to resolve conflicts using the isapnptools that we will discuss next, or you may have to reassign some of the ports or IRQs on non-PnP devices in order to get a working system.

Prior to the 2.4 kernel, a package called *isapnptools* allows a user to configure PnP devices. The `isapnp` command interprets a configuration file (normally */etc/isapnp.conf*) to configure PnP devices. This is usually done during the Linux boot process. The `pnpdump` command scans PnP devices and dumps a list of resources your PnP cards either need or would prefer to use. The format is suitable for use by the isapnp command, once you uncomment the actual commands that you wish to use. You must be sure to avoid resource conflicts. Refer to the man pages for isapnp and pnpdump for more information on using these commands.

Since the 2.4 kernel, PnP support has been integrated into the Linux kernel and the isapnptools package has become obsolete. For example, it was removed from Red Hat 7.3 which was released in May 2002. The support is similar to the PCI support discussed earlier. You can use the *lspnp* command (part of the kernel-pcmcia-cs package) to display information about PnP devices. You will also find this information in the /proc file system if the BIOS found PnP devices during initialization. The file /proc/bus/pnp will contain this information. This file will not be present on a PCI-only system.

IDE Hard drives

On modern PC systems, *Integrated Drive Electronics* or *IDE* hard drives are the most common. These are also known as *AT Attachment* or *ATA* drives after the original IBM PC-AT. Another type of drive using the *Small Computer System Interface* or *SCSI* interface is also popular, particularly on server machines. IDE drives have an advantage of low cost, while the SCSI interface permits attachment of a larger number of drives, with higher potential for overlapping operations to different drives on the same bus, and therefore higher potential performance.

A new type of drive, called *Serial ATA* or *SATA* has recently entered the market. The SATA specification seeks to address some of the limitations of the ATA specification while preserving significant compatibility with ATA.

BIOS and IDE drive sizes

IDE drives are formatted into *sectors*, data units of 512 bytes. A drive might contain multiple rotating disk platters, so the sectors are arranged in concentric circles with each circle called a *cylinder*. Data from a particular platter is read or written by a *head*. To find the data in a particular sector, the disk moves the head assembly to the cylinder, selects the appropriate head and waits for the right sector to come under the head. This gives rise to the notion of *CHS* (for Cylinder, Head and Sector) addressing. You may also hear this called *disk geometry*.

Unfortunately for history, early BIOS implemented a limit to the size permitted for each of the C, H and S values and DOS, a popular operating system for the PC, implemented a different limitation. During the 1990s, Disk sizes quickly outstripped the artificial CHS limitations imposed by BIOS and DOS. Several intermediate strategies involved translating the real CHS values to "virtual" values that would meet the constraints, either in the BIOS itself or by means of low level software routines such as Ontrack's Disk Manager software.

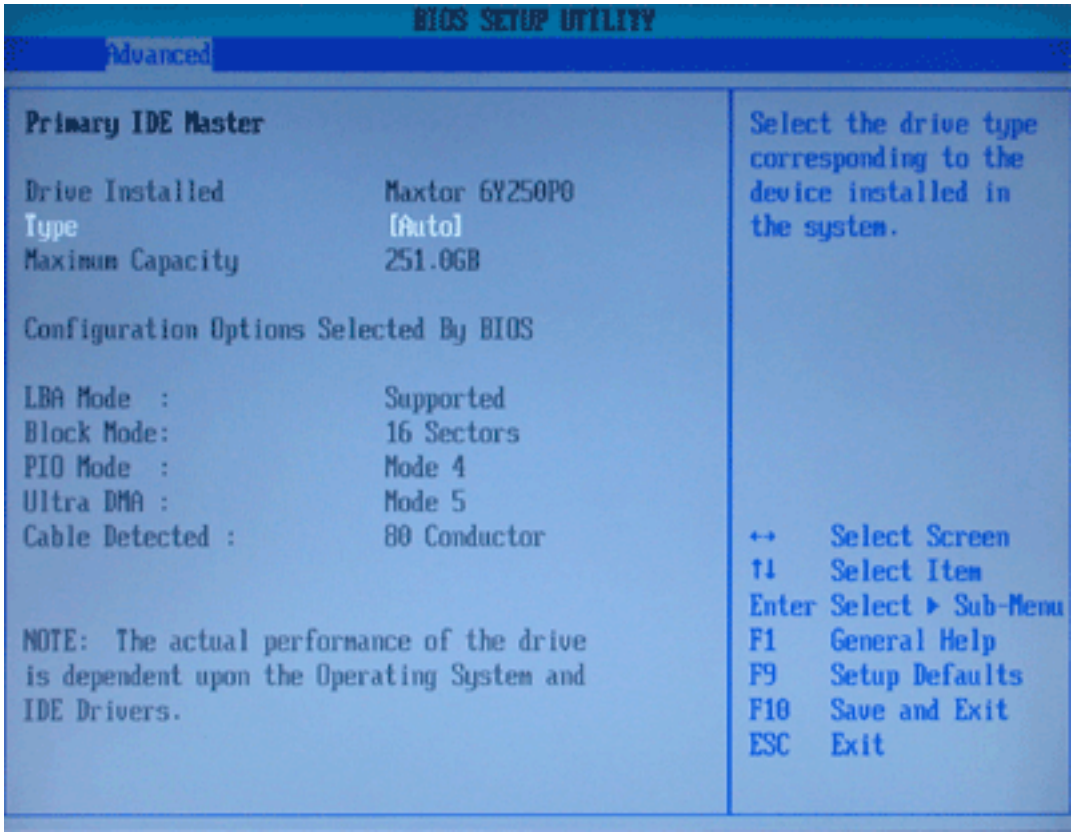
Even without the artificial limits of BIOS or DOS, the CHS design allows for up to 65536 cylinders, 16 heads, and 255 sectors/track. This limits the capacity to 267386880 sectors, or approximately 137 GB. Note that disk capacities, unlike some other PC values, are measured in powers of 10, so 1GB=1,000,000,000 bytes.

The solution was to have the system ignore the geometry and leave that to the drive to figure out. The system, instead of asking for a CHS value simply asks for a *Logical Block Address* or *LBA* and the drive electronics figure out which real sector to read or write. The process was standardized in 1996 with the adoption of the ATA-2 standard (ANSI standard X3.279-1996, *AT Attachment Interface with Extensions*).

As we discussed earlier, BIOS is needed to boot a system, so booting from a hard drive requires that the BIOS understand enough of the disk layout to locate and load the initial program that will then load the full operating system. An older BIOS that does not understand LBA disks will probably be limited to booting from within the first 1024 cylinders of a disk, or at least the first 1024 cylinders as the BIOS understands the disk geometry! Such a BIOS is probably now fairly rare, but if you do need to work with one, it may have a setting for LBA support and you may need to locate the /boot directory in a partition within the first 1024 cylinders. Even when your system will happily boot from the very end of a very large disk, many Linux partitioning tools will warn you that a partition extends beyond the 1024 cylinder limit.

Figure 3 shows information available in the BIOS of my Intel motherboard for the 250GB IDE disk on one of my Linux systems.

Figure 3. BIOS view of a large LBA disk



Listing 9 shows part of the output available on a Linux system (Fedora Core 3 in this case) using the `hdparm -I /dev/hda` command for the same disk as was used in Figure 3. Note that CHS values limit addressing to 4,128,705 sectors and the LBA value is set to 268,435,455 sectors or 137GB. These values together imply that the real capacity is in the LBA48 value. This is 490,234,752 sectors or 251GB.

Listing 9. Output from hdparm -I /dev/hda

```
/dev/hda:

ATA device, with non-removable media
        Model Number:       Maxtor 6Y250P0
        Serial Number:      Y638VBWE
        Firmware Revision:  YAR41BW0
Standards:
        Supported: 7 6 5 4
        Likely used: 7
Configuration:
        Logical         max     current
        cylinders       16383   65535
        heads           16      1
        sectors/track   63      63
        --
        CHS current addressable sectors:    4128705
        LBA    user addressable sectors:  268435455
        LBA48  user addressable sectors:  490234752
        device size with M = 1024*1024:      239372 MBytes
        device size with M = 1000*1000:      251000 MBytes (251 GB)
Capabilities:
        LBA, IORDY(can be disabled)
        Queue depth: 1
        ...
```

While we are discussing booting, one other point should be noted. By default, a PC will boot from the first IDE drive in the system. Some systems have BIOS settings that will allow you to override this, but most will boot this way. The system will first load a small piece of code from the *master boot record* and that will, in turn, provide information on which partition to boot. We will cover more about boot loaders for Linux in a later tutorial.

If you'd like to know even more about the history of large disks, see Resources for a link to the *Large Disk HOWTO* which is available from the Linux Documentation Project.

Linux disk names

We will cover a lot more about how Linux uses disks in later tutorials in this series. However, right now is a good time to introduce you to another important Linux file system, the */dev* filesystem. This, like /proc, is a pseudo file system which describes the devices that are or could be on a Linux system. Within the /dev filesystem you will find entries such as /dev/hda, /dev/hda5, /dev/sda, /dev/sdb1 and so on. You will find lots of other entries for other device types, but for now lets look at the ones that start with either */dev/hd* or */dev/sd*.

Devices that start with */dev/hd*, such as /dev/hda or /dev/hda5 refer to IDE drives. The first drive on the first IDE controller is /dev/hda and the second one, if present, is /dev/hdb. Likewise, the first drive on the second IDE controller is /dev/hdc and the

second one is /dev/hdd. As you can see from Listing 10, there are many more defined in /dev than are likely on your system.

Listing 10. /dev/hd? and /dev/sd? entries

```
[ian@lyrebird ian]$ ls /dev/hd?
/dev/hda  /dev/hdd  /dev/hdg  /dev/hdj  /dev/hdm  /dev/hdp  /dev/hds
/dev/hdb  /dev/hde  /dev/hdh  /dev/hdk  /dev/hdn  /dev/hdq  /dev/hdt
/dev/hdc  /dev/hdf  /dev/hdi  /dev/hdl  /dev/hdo  /dev/hdr
[ian@lyrebird ian]$ ls /dev/sd?
/dev/sda  /dev/sde  /dev/sdi  /dev/sdm  /dev/sdq  /dev/sdu  /dev/sdy
/dev/sdb  /dev/sdf  /dev/sdj  /dev/sdn  /dev/sdr  /dev/sdv  /dev/sdz
/dev/sdc  /dev/sdg  /dev/sdk  /dev/sdo  /dev/sds  /dev/sdw
/dev/sdd  /dev/sdh  /dev/sdl  /dev/sdp  /dev/sdt  /dev/sdx
```

As we did earlier for IRQs, we can use the dmesg command to find out what disk devices were found during bootstrap, Output from one of my systems is shown in Listing 11.

Listing 11. Hard drives found during bootup

```
[ian@lyrebird ian]$ dmesg | grep "[hs]d[a-z]"
Kernel command line: ro root=LABEL=RHEL3 hdd=ide-scsi
ide_setup: hdd=ide-scsi
    ide0: BM-DMA at 0x1860-0x1867, BIOS settings: hda:DMA, hdb:pio
    ide1: BM-DMA at 0x1868-0x186f, BIOS settings: hdc:DMA, hdd:DMA
hda: WDC WD1600JB-00EVA0, ATA DISK drive
hdc: Maxtor 6Y200P0, ATA DISK drive
hdd: SONY DVD RW DRU-700A, ATAPI CD/DVD-ROM drive
hda: attached ide-disk driver.
hda: host protected area => 1
hda: 312581808 sectors (160042 MB) w/8192KiB Cache,
     CHS=19457/255/63, UDMA(100)
hdc: attached ide-disk driver.
hdc: host protected area => 1
hdc: 398297088 sectors (203928 MB) w/7936KiB Cache,
     CHS=24792/255/63, UDMA(33)
 hda: hda1 hda2 hda3 hda4 < hda5 hda6 hda7 hda8 hda9 hda10 hda11 >
 hdc: hdc1 < hdc5 hdc6 hdc7 hdc8 >
hdd: attached ide-scsi driver.
```

From the highlighted lines in Listing 11, we see that the system has two IDE drives (hda and hdc) and a DVD-RW drive (hdd). Note that there is no hdb, indicating that there is no second drive on the first IDE controller on this system. An IDE drive can have up to four *primary* partitions and an unlimited number of *logical* partitions. Considering the drive hdc in Listing 11, we see that it has one primary partition (hdc1) and four logical partitions (hdc5, hdc6, hdc7, and hdc8). We will see in Topic 104 in a later tutorial in this series that hdc1 is actually a container (or *extended* partition) for the logical partitions.

Historically, devices such as sda and sdb were SCSI disks, which we will discuss further when we see how to set up SCSI devices Up to the 2.4 kernel, IDE CD and DVD devices were usually handled through SCSI emulation. Such a device often appeared in /dev as something like /dev/cdrom which was a symbolic link to the SCSI emulated device. For the above system, Listing 12 shows that /dev/cdrom is a link to /dev/scd0 rather than to /dev/hdd as might have been expected. Note the hdd=ide-scsi kernel parameter in Listing 11 as well as the indication that the ide-scsi

driver was attached for hdd.

Listing 12. IDE SCSI emulation

```
[ian@lyrebird ian]$ ls -l /dev/cdrom
lrwxrwxrwx  1 root   root   9 Jan 11 17:15 /dev/cdrom -> /dev/scd0
```

Today, you will find that both USB and SATA storage devices appear as sd, rather than hd, devices.

Legacy peripherals

We have alluded above to peripherals such as serial or parallel ports that are usually integrated into a motherboard, and we have seen some standard IO port and IRQ assignments for these devices. Serial ports, in particular, have been used for connecting a variety of devices and they have a history of being hard to configure. With the advent of *IEEE 1394*, also known as *Firewire* and *Universal Serial Bus* or *USB* devices, automatic configuration and hot plugging of devices has largely replaced the chore of ensuring correct serial or parallel port configuration. Indeed, a *legacy-free* system does not support the standard serial or parallel ports. Neither does it support a floppy drive or a PS/2 connected keyboard or mouse.

We'll now discuss some common BIOS settings that you may need to configure.

Serial ports (COMn)

The legacy serial ports are known as COM1 through COM4. If your system has a single serial port connector (originally a 25-pin DB25 connector but now more commonly a 9-pin DB9 connector) it will probably use the default base address and IRQ for COM1, namely IO port 3F8 and IRQ 4. The standard IO port addresses and IRQs for serial ports are shown in Table 3.

Table 3. Serial port assignments

Name	Address	IRQ
COM1	3F8-3FF	4
COM2	2F8-2FF	3
COM3	3E8-3EF	4
COM4	2E8-2EF	3

You will notice that COM1 and COM3 share IRQ 4 and likewise COM2 and COM4 share IRQ 3. Unless the driver and the device can actually share the interrupt, or a device does not use interrupts, this means that most real systems will use only COM1 and COM2.

Occasionally, you may need to either disable an onboard serial port or configure it to use an alternate address and IRQ. The most likely reason to do this is because of

conflicts with a PnP modem in an ISA slot or a desire to use the PnP modem as COM1. We recommend that you only change these if you are having problems with Linux detecting your configuration.

Parallel ports (LPTn)

The legacy parallel ports are known as LPT1 through LPT4, although usually only at most two are present. If your system has a single parallel port connector it will probably use the default base address and IRQ for LPT1, namely IO port 378 and IRQ 7. The standard IO port addresses and IRQs for parallel ports are shown in Table 4.

Table 4. Parallel port assignments

Name	Address	IRQ
LPT1	378-37F	7
LPT2	278-27F	5
LPT*	3BC-3BE	

Note that the IO ports 3BC-3BE were originally used on a Hercules graphics adapter that also had a parallel port. Many BIOS systems will assign this range to LPT1 and then the other two ranges would become LPT2 and LPT3 respectively instead of LPT1 and LPT2.

Many systems do not use interrupts for printers, so the IRQ may or may not actually be used. It is also not uncommon to share IRQs for printing and also to share IRQ 7 with a sound card (Sound Blaster compatible).

The parallel ports were originally used for printing with data flowing to the printer and a few lines reserved for reporting status. Later, the parallel port was used for attaching a variety of devices (including early CD-ROMs and tape drives), so the output-only nature of the data flow changed to a bidirectional data flow.

The current standard applicable to parallel ports is *IEEE Std. 1284-1994 Standard Signaling Method for a Bi-Directional Parallel Peripheral Interface for Personal Computers* which defines five signaling modes. Your BIOS may give you choices in setup such as *bi-directional*, *EPP*, *ECP* and *EPP and ECP*. ECP stands for *Enhanced Capabilities Port* and is designed for use with printers. EPP stands for *Enhanced Parallel Port* and is designed for devices such as CD-ROMs and Tape drives which require large amounts of data to flow in either direction. The default BIOS choice is likely to be ECP. As for serial ports, change this only when you have a device that does not work properly.

Floppy disk port

If your system has a legacy floppy disk controller, it will use ports 3F0-3F7. If you install a legacy floppy drive in a system that shipped without one, you may have to enable legacy options in your BIOS. Consult the manufacturer's information for more details.

Keyboard and mouse

The keyboard/mouse controller uses ports 0060 and 0064 for legacy keyboards and mice. That is, those connected by a round PS2 connector. Many systems will generate a Power-On-Self-Test (POST) error if a keyboard is not attached. Most machines designed to be used as servers, and many desktops, now have BIOS options to allow clean startup without a keyboard or mouse present.

Once a system is installed, running without a keyboard (or mouse) is seldom a problem. Servers frequently run this way. Management is performed over the network using either web administration tools, or a command line interface such as telnet or (preferably) ssh.

Installation on a keyboardless system is usually accomplished using a terminal (or terminal emulator) attached through a serial port. Usually, you will need a keyboard and display to ensure that the BIOS is set up correctly with an enabled serial port. You may also need a customized boot disk or CD to perform a Linux system install.

Another approach used by systems such as the IBM JS20 blade server is to emulate a serial connection over a LAN.

Section 3. Modems and sound cards

This section covers material for topic 1.101.3 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

Modems

A *modem* (from *mod*ulator/*dem*odulator) is a device for converting the digital signals used in computers to a serial stream of analog data that is transmitted over telephone lines. In the early days of PCs, modems were external devices that were attached to a serial port. Later, modems were implemented on cards that could be installed inside the computer, reducing cost for housing and power, and eliminating the need for a cable between serial port and modem. Another cost reduction occurred when some of the function normally done by a modem was transferred to software in the PC. This type of modem may be called a softmodem, HCF modem, HSP modem, HSF modem or controllerless modem, among other terms. Such modems were designed to reduce the cost of systems which generally ran Microsoft Windows. The term *winmodem* is often used for such devices, although Winmodem® is a registered trademark of U.S. Robotics, who manufactured several modems under that name.

Most external modems and full function internal modems will work under Linux

without problem. Some of the modems that require software assistance from the PC operating system will also work with Linux and the list of working modems in this category is continually increasing. Software-assisted modems that work under Linux are often called *linmodems* and there is a site dedicated to these (linmodems.org). If you have such a modem, you first step should be to check the linmodems site (see Resources) and download the latest version of the scanModem tool. This will tell you what is already known about available drivers (if any) for your modem.

If you have an ISA modem, you will need to ensure that ports, IRQs and DMA channels do not conflict with other devices. See the earlier section BIOS settings for additional information.

The modems discussed in this section are *asynchronous* modems. There is another class of modems, called *synchronous* modems used for HDLC, SDLC, BSC or ISDN. Very loosely, we can say that asynchronous transmission is concerned with transmitting individual bytes of information while synchronous communications is concerned with transmitting whole blocks of information.

Most Linux communications occurs using the *Internet Protocol* or *IP* So a Linux system will need to run what looks like IP over an asynchronous line which was not originally designed for block protocols such as IP. The first method of doing this was called *Serial Line Interface Protocol* or *SLIP*. A variant using compressed headers is called *CSLIP*. Nowadays, most Internet Service Providers (ISPs) support dialup connections using *Point-to-Point Protocol* or *PPP*.

The *Linux Networking-HOWTO* and *The Network Administrators' Guide* available from the Linux Documentation Project (see Resources) provide information on SLIP, CSLIP and PPP configuration.

When communicating using a modem, there are a number of settings that you may need to make on your Linux system. Most importantly, you will set the speed of communications between your system and the modem. this will usually be higher than the nominal line speed and is usually set to the maximum supported by your serial port chipset and your modem. One way to set or view the modem parameters that will be used by the serial driver is with the `setserial` program. We illustrate the setserial command in Listing 13. Note that the `-G` option prints the output in a format suitable for use in setting parameters with setserial. In this case, the UART (Universal Asynchronous Receiver Transmitter) is a buffered 16550 which is a common type of UART on modern PCs. The speed is set of 115,200 bps which is also commonly used with this UART and most modern external 56kbps modems. Note that the default speed on some newer systems may be set as high as 460,800bps. If your modem does not appear to respond, this is probably the first thing you should check.

Listing 13. The setserial command

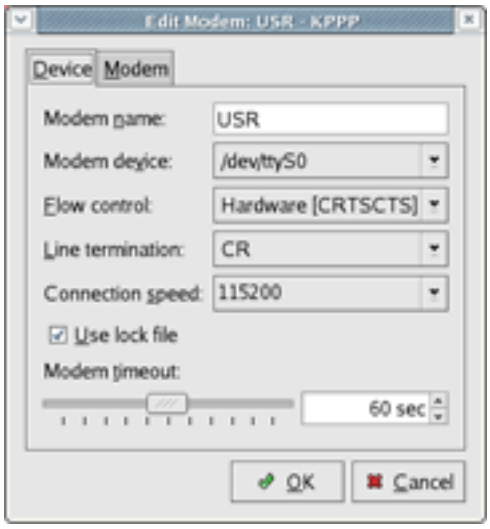
```
[root@attic4 ~]# setserial /dev/ttyS0
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
[root@attic4 ~]# setserial -G /dev/ttyS0
/dev/ttyS0 uart 16550A port 0x03f8 irq 4 baud_base 115200 spd_normal skip_test
```

One thing to note about setserial is that it does not probe the hardware. All it does is tell the serial driver what parameters to use, unless you use the autoconfig and auto_irq parameters. In this case, setserial will ask the kernel to probe the hardware. See the man pages for setserial for more information about these and other options of the command.

We will cover networking more in a tutorial for LPI exam 102 (See Resources). In the meantime, if you wish to set up a PPP connection, there are several excellent tools to help you do this. The `kppp` program has a nice GUI and is easy to use. The `wvdial` command provides an intelligent command line tool for setting up dial connections. In addition to these, distributions may have other tools, either specifically for PPP or dialup connections or as part of a more general network configuration tool such as `system-config-network` in Fedora Core 4.

Another aspect of modem communications that is usually under control of the communications program but may be set or have the default values set on the modem itself is *flow control*. This is a way for one end to tell the other end to wait for a moment while the receiving end clears its data buffers. This may be done in software by sending XON and XOFF characters. The preferred way, and that used for PPP connections, is called *hardware flow control* in which the state of certain modem signal lines is used to indicate readiness to receive data. The signals used are *Clear to Send* or *CTS* and *Ready to Send* or *RTS*, so you will often see this described as flow control using RTS/CTS or something similar. Figure 4 shows how the speed and hardware flow control are set using the kppp program.

Figure 4. Configuring modem parameters with kppp



Sound cards

Most personal computers sold today include audio or *sound card* capabilities.

Sound port (Sound Blaster)

The Creative Labs Sound Blaster series of sound cards have set de facto industry standards for sound cards. Even though many other brands of excellent sound cards

exist, many of these provide a compatibility mode for one or more of the Sound Blaster series. The original Sound Blaster card was an 8-bit card that worked in the original IBM PC. Later 16-bit models for the PC-AT and compatibles used the 16-bit PC-AT or ISA bus. Today, most of these cards use the PCI bus. Many motherboards even provide a sound chip with Sound Blaster compatibility on board. Sound devices may also be attached through USB connections, although we will not cover those here.

The ports used by an ISA bus Sound Blaster card are 0220-022F, although base addresses of 240, 260 or 280 were often configurable. Similarly, the IRQ is usually configurable, with common choices being 2, 5, 7, or 10. The default setting is to use IRQ 5. The cards could usually be configured to use alternate DMA channels too.

As with all ISA devices, you will need to ensure that ports, IRQs and DMA channels do not conflict with other devices. See the earlier section BIOS settings for additional information.

MIDI port (MPU-401)

Many sound cards also have in interface to attach a *MIDI* (from Musical Instrument Digital Interface) device. Commonly, this interface emulates the Roland MPU-401. The standard ports used by the MPU-401 ISA interface are 0200-020F.

As with all ISA devices, you will need to ensure that ports, IRQs and DMA channels do not conflict with other devices. See the earlier section BIOS settings for additional information.

Configuring Linux sound support

Modern 2.4 and 2.6 kernels have sound support for a wide variety of sound devices built in to the kernel, usually as modules. As with other devices, we can use the `pnpdump` command for ISA devices, or the `lspci` command for PCI devices to display information about the device. Listing 14 shows the output from lspci for an Intel sound system on a system motherboard.

Listing 14. Using lspci to display sound resources

```
[root@lyrebird root]# lspci | grep aud
00:1f.5 Multimedia audio controller: Intel Corporation 82801DB/DBL/DBM
               (ICH4/ICH4-L/ICH4-M) AC'97 Audio Controller (rev 01)
```

Kernel modules are the preferred way to provide support for a variety of devices. Modules need only be loaded for the devices actually present and they may be unloaded and reloaded without rebooting the Linux system. For 2.4 and earlier kernels, the module configuration information is stored in /etc/modules.conf. For 2.6 kernels, the kernel module system was redesigned and the information is now stored in /etc/modprobe.conf. In either case, the `lsmod` command will format the contents of /proc/modules and display the status of loaded modules.

Listing 15 shows the contents of /etc/modprobe.conf for a 2.6 kernel and Listing 16

shows the output from lsmod as it relates to sound devices on this system.

Listing 15. Sample /etc/modprobe.conf (2.6 kernel)

```
[root@attic4 ~]# cat /etc/modprobe.conf
alias eth0 e100
alias snd-card-0 snd-intel8x0
install snd-intel8x0 /sbin/modprobe --ignore-install snd-intel8x0 &&\
 /usr/sbin/alsactl restore >/dev/null 2>&1 || :
remove snd-intel8x0 { /usr/sbin/alsactl store >/dev/null 2>&1 || : ; }; \
/sbin/modprobe -r --ignore-remove snd-intel8x0
alias usb-controller ehci-hcd
alias usb-controller1 uhci-hcd
```

Listing 16. Sound related output from lsmod (2.6 kernel)

```
[root@attic4 ~]# lsmod |egrep '(snd)|(Module)'
Module                  Size  Used by
snd_intel8x0           34689  1
snd_ac97_codec         75961  1 snd_intel8x0
snd_seq_dummy           3653  0
snd_seq_oss            37057  0
snd_seq_midi_event      9153  1 snd_seq_oss
snd_seq                62289  5 snd_seq_dummy,snd_seq_oss,snd_seq_midi_event
snd_seq_device          8781  3 snd_seq_dummy,snd_seq_oss,snd_seq
snd_pcm_oss            51185  0
snd_mixer_oss          17857  1 snd_pcm_oss
snd_pcm               100169  3 snd_intel8x0,snd_ac97_codec,snd_pcm_oss
snd_timer              33605  2 snd_seq,snd_pcm
snd                    57157  11 snd_intel8x0,snd_ac97_codec,snd_seq_oss,
  snd_seq,snd_seq_device,snd_pcm_oss,snd_mixer_oss,snd_pcm,snd_timer
soundcore              10913  1 snd
snd_page_alloc          9669  2 snd_intel8x0,snd_pcm
```

Listing 17 shows the contents of /etc/modules.conf for a 2.4 kernel and Listing 18 shows the output from lsmod as it relates to sound devices on this system. Note the similarities between the modules.conf and modprobe.conf files.

Listing 17. Sample /etc/modules.conf (2.4 kernel)

```
[root@lyrebird root]# cat /etc/modules.conf
alias eth0 e100
alias usb-controller usb-uhci
alias usb-controller1 ehci-hcd
alias sound-slot-0 i810_audio
post-install sound-slot-0 /bin/aumix-minimal -f /etc/.aumixrc -L >/dev/null 2>&1 || :
pre-remove sound-slot-0 /bin/aumix-minimal -f /etc/.aumixrc -S >/dev/null 2>&1 || :
```

Listing 18. Sound related output from lsmod (2.4 kernel)

```
Module                  Size  Used by    Not tainted
smbfs                  43568   1  (autoclean)
i810_audio             28824   0  (autoclean)
ac97_codec             16840   0  (autoclean) [i810_audio]
soundcore               6436   2  (autoclean) [i810_audio]
st                     30788   0  (autoclean) (unused)
```

Sound support on many 2.4 and earlier systems is provided through the *Open Sound System (OSS) Free* drivers. Many systems today use the *Advanced Linux*

sound architecture or *ALSA* drivers. The `sndconfig` utility was created by Red Hat to assist in configuring ISA PnP sound cards. It also works with PCI sound cards. This utility may be present on systems that do not use the ALSA drivers, although modern module support has made it largely unnecessary. The utility will probe for sound cards, lay a test sound of Linus Torvalds speaking, and then update the /etc/modules.conf file. Typical operation is shown in Figures 5 and 6.

Figure 5. The sndconfig utility

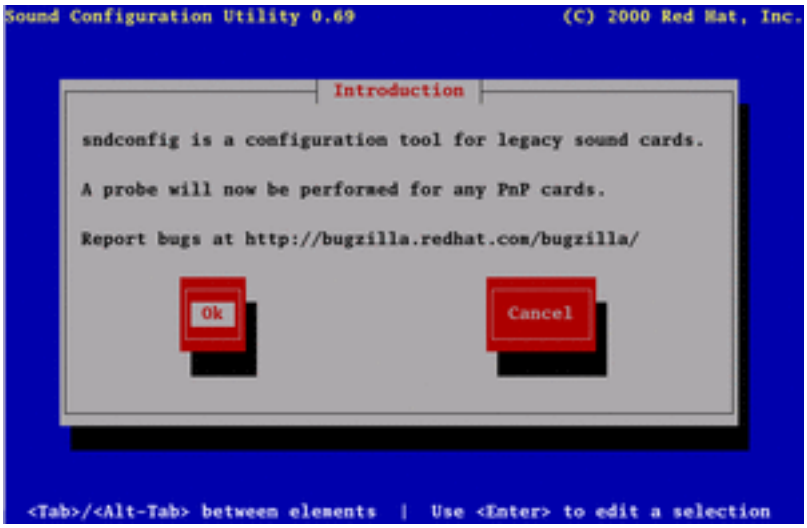
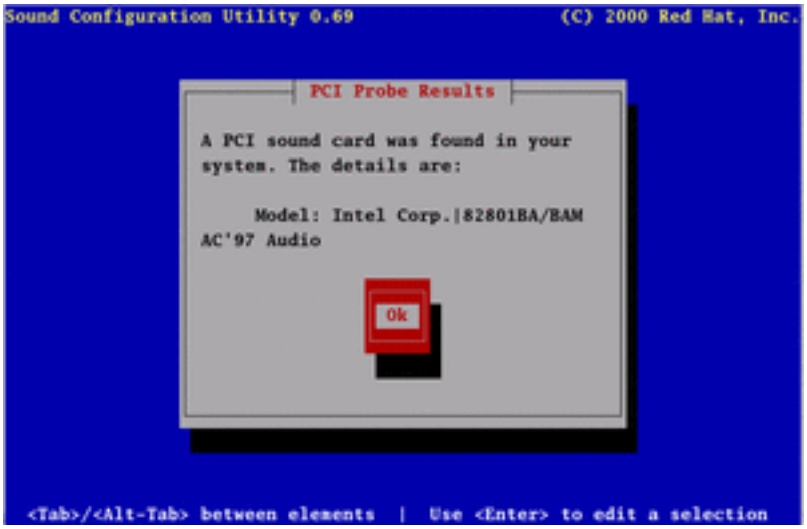


Figure 6. The sndconfig utility



Section 4. Set up SCSI devices

This section covers material for topic 1.101.4 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

SCSI overview

The *Small Computer System Interface*, more generally known as *SCSI*, is an interface designed for connecting streaming devices such as tapes and block storage devices such as disks, CD-ROMs, and DVDs. It has also been used for other devices, such as scanners and printers. SCSI is pronounced "scuzzy". SCSI was designed to allow multiple devices on the bus. One device, called the *controller* has responsibility for managing the bus. SCSI devices may be either internal or external.

There have been three major releases of SCSI standards from the American National Standards Institute (ANSI).

SCSI
is the original standard (X3.131-1986), now usually called *SCSI-1*. This arose from efforts by Shugart Associates to get a standard interface for disk devices. The standard supported up to 8 devices on a cable. SCSI-1 uses passive termination (more on this below). This standard has now been withdrawn, although devices may still work on current SCSI cables assuming appropriate termination. The data interface was 8 bits parallel with a maximum speed of 5 MBps (megabytes/sec). The SCSI standard was designed for disks, but is very flexible and was used for other devices, notably scanners and slower devices such as Zip. FConnection used a 50 connector cable, originally with a Centronics connector, but later with a 50-pin D-shell connector, similar to a DB-25 RS-232 serial connector,

SCSI-2
was approved as ANSI standard X3.131-1994 in 1994. This revision doubled the speed of the bus to 10MBps as well as introducing so-called *wide* or 16-bit data transfers. A 16-bit bus running at 10MBps can transfer 20MBps of data. The 50-connector cable was used for 8-bit or *narrow* SCSI devices, while the newer wide devices used a 68-pin cable. Higher density cables were also introduced, allowing smaller and cheaper connectors. SCSI-2 also standardized the SCSI command set and introduced differential signaling to improve quality at higher speeds. This was later called *High Voltage Differential* or *HVD* signaling. HVD has active termination requirements. It is possible to mix 8-bit and 16-bit devices on a cable with appropriate care in termination. SCSI-2 supports up to 16 devices on a cable of which at most 8 may be narrow.

SCSI-3
is a set of standards rather than a single standard. This allows standards to be enhanced for technology areas that are fast-moving, while avoiding the need to revise standards for stable technology. The overall architecture is defined in ANSI standard X3.270-1996 which is also known as the *SCSI-3 Architecture Model* or *SAM*. The earlier SCSI standards are now embodied in the *SCSI Parallel Interface* or *SPI* standards. Speed was increased again and current 16-bit devices are capable of up to 320MBps data transfers at a bus speed of 160MBps.
SCSI-3 introduced Fiber Channel SCSI with support for up to 126 devices per bus allowing connection over 1GBps or 2GBps fiber channel links at distances up to several kilometers. This helps to alleviate inherent limitations involved

with the use of standard SCSI cabling. Another notable introduction was *Single Connector Attachment* or *SCA* which is only used for wide (16-bit) devices. SCA is an 80-pin connector which incorporates the pins from the 68-pin connector as well as power and some additional pins. SCA is designed to allow devices to be safely hot-plugged in a running system, and is frequently used in devices implementing *Redundant Array of Independent disks* or *RAID* storage systems as well as network attached storage and server racks.

We mentioned *termination* above without saying much about it. The electrical specifications for a SCSI bus require each end of the bus to be properly terminated. You must use the appropriate type of terminator for your bus; passive, HVD or LVD. If you mix wide and narrow devices on a bus be aware that the termination for narrow devices may occur in a different place to the termination for wide devices. If the controller is controlling only an internal bus or only an external bus, it will usually provide termination, either automatically or via BIOS configuration. Check the manuals for your particular controller. If the controller is controlling both an internal and an external segment, then it should normally not provide termination.

Some devices are capable of providing termination, either via a switch, or other means such as a jumper. Again, consult the manual for your device. Otherwise, termination is usually accomplished with a terminator block which is plugged into the cable. Whichever type of termination you use, be particularly careful if you mix wide and narrow devices on the same bus, as the narrow termination may occur at a different place on the cable than the wide termination.

SCSI Ids

By now, you may be wondering how the system manages many devices on one cable. Every device, including the controller, has an *ID*, represented by a number. For narrow (8-bit) SCSI, the ID numbers range from 0 through 7. Wide SCSI adds numbers 8 through 15. Narrow devices may only use ID numbers 0 through 7 while wide devices may use 0 through 15. The controller is generally assigned ID 7. The ID for a devices may be set via jumpers, switches or dials on the device, or through software. Devices using the Single Connector Attachment (SCA) usually have an ID assigned automatically as these devices may be hot-plugged.

Devices on a SCSI bus have a priority. Priority for narrow devices runs from 0 (lowest) through 7 (highest), so a controller at address 7 has highest priority. The extra IDS for wide SCSI have priority 8 (lowest) through 15) highest, with 15 having lower priority than 0. Thus, the overall priority sequence is 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7. Slower devices and devices that cannot tolerate delays (such as CD or DVD recorders) should be given high priority IDs to ensure they get sufficient service.

Devices such as RAID controllers may present a single ID to the bus but may incorporate several disks. In addition to the ID, the SCSI addressing allows a *Logical Unit Number* or *LUN*. Tapes and single disk drives either do not report a LUN or

report an LUN of 0.

A SCSI adapter may support more than one SCSI cable or *channel*, and there may be multiple SCSI adapters in a system. The full ID of a device therefore consists of an adapter number, a channel number, a device ID and a LUN.

Devices such as CD recorders using ide-scsi emulation and USB storage devices will also appear to have their own adapter.

Linux names and files for SCSI devices

Back in the BIOS section on IDE drives we discussed the names assigned by Linux to the various IDE devices, such as /dev/hda and /dev/hdc. This is simple for an IDE controller which can support either one or two hard drives. The secondary IDE drive on the second adapter is always /dev/hdd, even if the only other hard drive is the primary drive on the first adapter (/dev/hda). With SCSI the situation becomes more complicated as we may mix hard drives, tapes, CD and DVD drives, as well as other devices on a SCSI cable.

Linux will assign device names as devices are detected during boot. Thus, the first hard drive on the first channel of the first adapter will become /dev/sda, the second /dev/sdb, and so on. The first tape drive will be /dev/st0, the second /dev/st1, and so on. The first CD device will become /dev/sr0 or /dev/scd0 and the second /dev/sr1 or /dev/scd1. Devices using SCSI emulation, such as USB storage devices and (prior to the 2.6 kernel) IDE CD or DVD drives will also be allocated names in this name space.

While we won't cover all the intricacies of SCSI naming here, it is most important to know that this numbering is redone at each boot. If you add or remove a SCSI hard drive, then all previously higher drives will have a different device name next time you boot. The same goes for other device types. We will learn more about partitions, labels and file systems in another tutorial in this series, but for now we will warn you about one thing. Since disks can have up to 15 partitions on them, each with a name tied to the device name (for example, /dev/sda1, /dev/sda2 through /dev/sda15), this can cause havoc when your system attempts to mount the filesystems. Plan very carefully when you add or remove SCSI devices and use disk labels rather than device names for SCSI disks whenever possible.

We introduced the /proc file system in the section on BIOS settings. The /proc file system also contains information about SCSI devices. Listing 19 shows the contents of /proc/scsi/scsi on a system with two SCSI devices, a hard drive with ID 0 and a controller with ID 8.

Listing 19. /proc/scsi/scsi

```
[root@waratah root]# cat /proc/scsi/scsi
Attached devices:
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: IBM-PSG  Model: DPSS-336950M  F  Rev: S94S
  Type:   Direct-Access                    ANSI SCSI revision: 03
Host: scsi1 Channel: 00 Id: 08 Lun: 00
  Vendor: IBM      Model: YGLv3 S2         Rev: 0
```

```
  Type:   Processor                        ANSI SCSI revision: 02
```

If you want to know which real device corresponds to say /dev/sda, you can use the `scsi_info` command. Listing 20 confirms that our first (and only) SCSI hard drive is /dev/sda.

Listing 20. The scsi_info command

```
[root@waratah root]# scsi_info /dev/sda
SCSI_ID="0,0,0"
MODEL="IBM-PSG DPSS-336950M  F"
FW_REV="S94S"
```

However, note that some systems, such as Fedora Core 2, do not include the scsi_info command (which is a part of the kernel-pcmcia-cs package).

More recent systems have switched to using the *SCSI Generic* or *sg* driver. When the sg driver is used, you will find additional information under the /proc/scsi/sg subtree in your filesystem. You will also have devices such as /dev/sg0, /dev/sg1. /dev/sg2 and so on. These generic devices usually correspond to some other device type such as a hard disk like /dev/sda or a tape like /dev/st0.

The sg3_utils package contains a number of utilities for manipulating and interrogating aspects of the SCSI subsystem. In particular, the `sg_map` command will provide a map between the sg name and another device name if one exists. Note that scanners will not have another device name, only a generic one. Listing 21 shows the output of sg_map on a system with an IDE optical drive that uses SCSI emulation and two USB drives.

Listing 21. The sg_map command

```
[root@lyrebird root]# sg_map
/dev/sg0  /dev/scd0
/dev/sg1  /dev/sda
/dev/sg2  /dev/sdb
```

The sg utility corresponding to scsi_info is `sginfo`. You can use either the generic device name or the more familiar name with sginfo. Listing 22 shows the output of sginfo for the three devices of Listing 21. Notice that sginfo does not provide information for /dev/sg1, although as shown in the listing the scsi_info command does show it as a USB memory key. In this case, the device has been unplugged from the system. Information about it is retained (and can be found in /proc/scsi/scsi). the sginfo command interrogates the device for the information while the scsi_info will use the retained information. Thus sginfo must be run as root while scsi_info need not be run as root, although non root users may have to specify the full path of /sbin/scsi_info.

Listing 22. The sginfo command

```
[root@lyrebird root]# sginfo /dev/scd0
```

```
INQUIRY response (cmd: 0x12)
----------------------------
Device Type                          5
Vendor:                     SONY
Product:                    DVD RW DRU-700A
Revision level:             VY08

[root@lyrebird root]# sginfo /dev/sg1
INQUIRY reponse (cmd: 0x12)
----------------------------
Device Type                          0
Vendor:
Product:
Revision level:

[root@lyrebird root]# sginfo /dev/sg2
INQUIRY reponse (cmd: 0x12)
----------------------------
Device Type                          0
Vendor:                     WD
Product:                    2500JB External
Revision level:             0411

[root@lyrebird root]# scsi_info /dev/sg1
SCSI_ID="0,0,0"
MODEL=" USB DISK 12X"
FW_REV="2.00"
```

SCSI BIOS and boot sequence

While SCSI is standard on most servers, most desktop and laptop computers do not normally include SCSI support as standard. Such systems will normally boot from a floppy disk, a CD or DVD drive or the first IDE hard drive in the system. The boot order is usually configurable in BIOS setup screens such as we saw in the section BIOS settings, and sometimes dynamically by pressing a key or key combination during system startup.

The BIOS Boot Specification (see Resources) defines a method for add on cards such as SCSI cards to present a message during startup and have BIOS on the card invoked for configuration purposes. SCSI cards normally use this to allow configuration of the SCSI subsystem controlled by the card. For example, an Adaptec AHA-2930U2 card will present a message

```
Press <Ctrl><A> for SCSISelect (TM) Utility!
```

allowing a user to press the ctrl and A keys together to enter the adapter BIOS. Other cards will have a similar process for entering the card BIOS to set up the card.

Once in the card BIOS, you will have screens that typically allow you to set the SCSI controller address (typically 7), the SCSI boot device (usually ID 0), the bus speed and whether the controller should provide termination or not. Some older cards may require that the boot device be ID 0, but most modern cards will allow you to choose any device. You may, and probably will, have other options, such as the ability to format a hard disk. See your card manufacturer's documentation for details. Once you have set up the SCSI view of the bus, you will usually still have to tell your PC BIOS to boot from the SCSI disk rather than an IDE drive. Consult your system reference manual to determine whether you can boot from a non-IDE drive and how

to set it if you can.

Section 5. PC expansion cards

This section covers material for topic 1.101.5 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

We covered the material that you will need to know for this section when we discussed BIOS settings. You should review the discussion of DMA, IRQs, ports and the different kinds of buses and adapters in the section Buses, ports, IRQs, and DMA so you understand the contents of the /proc/dma, /proc/interrupts, and /proc/ioports files and how to use them to determine any conflicts. Review the material on /proc/pci and the lspci command. Also review the material in the Plug and play section for information about ISA and Plug and Play cards. There you will find information about isapnp and pnpdump.

Section 6. Communication devices

This section covers material for topic 1.101.6 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

This section covers a variety of communications devices, including modems, ISDN adapters, and DSL switches. This material for this section falls into two general categories:

1. Selecting and installing your communications device, and
2. Communicating with your device

Selecting your communications device is like selecting any other device for your system in that it needs to match your bus type (PCI or ISA) and you need Linux support for the device. You should review the discussion of DMA, IRQs, ports and the different kinds of buses and adapters in the section Buses, ports, IRQs, and DMA so you understand the contents of the /proc/dma, /proc/interrupts, and /proc/ioports files and how to use them to determine any conflicts. Review the material on /proc/pci and the lspci command. Also review the material in the Plug and play section for information about ISA and Plug and Play cards. There you will find information about isapnp and pnpdump.

The Linux kernel supports more and more devices with every release, so your first

check for support should be with the distribution you are already using. If the support is already installed, your distribution may have a utility to help you configure it. Figure 7 illustrates the Fedora Core 4 network configuration tool. You can see that an ethernet connection has been configured (and is active) and a backup dial connection using PPP has also been configured. The system already supports that addition of ISDN, Token Ring, wireless and xDSL connections.

Figure 7. Fedora Core network configuration utility

If you have to install drivers for a communications device, check first to see if the required drivers are a part of your distribution that has not yet been installed and install if so. Otherwise, you should try and find a driver package that has already been built for your system. Your final choice is to build your own driver package from source. We will cover building packages in the tutorial for LPI Exam 101 Topic 102. (see Resources).

For an ISDN connection, you will also need the synchronous PPP driver, as the normal one used with asynchronous modems is designed for character mode transmission rather than block mode. As we mentioned in the section on Modems we will discuss setting up connections more in a tutorial for LPI exam 102 (See Resources).

DSL connections may be one of several types. Some provide an ethernet port that is bridged to the ISP network. Authentication is usually done in this case using your

computer's ethernet MAC address. If you attach a router (or a different computer) to the DSL modem, you may need to clone the MAC address of the computer that was originally connected in order for the connection to work. More commonly, an ISP will use *Point-to-Point Protocol over Ethernet* or *PPPoE*. In this case, you are provided with a username and password to use when establishing the connection. In this case, if you use a router, you will usually configure this address into the router and your computer will simply use a standard ethernet connection. Rarely, you may have a *PPPoA* or *PPP over ATM* connection.

Wireless connections may require you to know the name of the network you are connecting to. This is called a *Service Set Identifier* or *SSID*. If the network uses encryption such as *Wired Equivalent Privacy* or *WEP* or *WiFi Protected Access* or *WPA* you will need to configure your connection appropriately.

Section 7. USB devices

This section covers material for topic 1.101.7 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

USB overview

In this section we will look at Linux support for *Universal Serial Bus* or *USB* devices. USB was developed by a consortium of companies with the goal of providing a single, simple bus for attachment of peripherals. In the section on BIOS settings, we saw the complexities of managing ports, IRQs and DMA resources in ISA bus machines. The USB design allows devices to be hot-plugged and uses standard connectors for connecting devices. USB devices include keyboards, mice, printers, scanners, hard drives, flash memory drives, cameras, modems, ethernet adapters, and speakers. The list keeps growing. Current Linux support is quite comprehensive, although some devices require special drivers and others, particularly printers, may not be supported or may be only partially supported.

A computer system may provide one or more *controllers* or *hubs.* to which either a USB device or another (external) hub may be connected. A hub can support up to 7 devices, some or all of which may be additional hubs. The hub in the system is called the *root hub*. Each such star topology can support up to 127 hubs or devices.

Note: Frequently, we speak of a *USB port* which refers to the USB capability in a computer and the connecting socket (compare with serial port or parallel port) rather than the internal port addresses used by the device.

The USB system is a layered system.

1. The *Bus Interface* layer provides physical, signaling, and packet connectivity between hosts and devices, providing data transfer between

the host and devices.

2. The *Device* layer is used by the system software to do generic USB operations with a device over the bus. This allows the host to determine characteristics of the device, including device class, vendor name, device name, power requirements, and many capabilities such as device speed or USB level supported.

3. The *Function* layer provides additional capabilities that are specific to the device. Matched host and device software layers permit use device-specific functions.

The earlier USB specifications (1.0 and 1.1) support speeds up to 12Mbps (mega*bits* per second). Devices conforming to this specification are relatively low speed devices, such as printers, mice, keyboards, scanners, and modems. The newer USB 2.0 specification supports speeds up to 480Mbps which is adequate for hard drives and external CD or DVD drives. Some USB 2.0 devices are backwards compatible to allow use on older systems, although not all faster devices are backwards compatible. If your computer does not have USB 2.0 support built in, PCI cards (or PC cards for laptops) are available to provide one or more USB 2.0 ports.

The USB cable is a thin, 4-wire cable with two signal lines plus power and ground. The end plugged into a hub has a flat rectangular connector (called an A connector) while the end plugged into a device or downstream hub has a small more square, connector (the B connector). Several different mini-B connectors exist for connecting small devices such as cameras to a computer. USB devices and hubs may draw power from the USB bus or may be self powered.

Linux USB module support

USB is now fairly well supported in Linux. Much of the development has occurred in the 2.6 kernel tree. A lot has been backported to 2.4 kernels, with some support even in 2.2 kernels. Linux supports USB 2.0 as well as the earlier specifications. Because of the hot-pluggable nature of USB, support is usually provided through kernel modules which can be loaded or unloaded as necessary. For this tutorial we will assume that the modules you need for your distribution are either available or already installed. If you need to compile your own kernel, refer to the tutorial for Exam 201 Topic 201 (see Resources).

After you have ascertained that your computer has USB ports, you may check what your Linux system found using the `lspci` command as shown in Listing 23. We have filtered the output to show just USB related devices.

Listing 23. lspci output for USB devices

```
[root@lyrebird root]# lspci | grep -i usb
00:1d.0 USB Controller: Intel Corporation 82801DB/DBL/DBM
     (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801DB/DBL/DBM
```

```
     (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801DB/DBL/DBM
     (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #3 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801DB/DBM
     (ICH4/ICH4-M) USB2 EHCI Controller (rev 01)
```

You will notice that there are four USB controllers in this system. The UHCI and EHCI fields indicate the driver module required to support the controller. The correct USB 1.1 driver depends on the chipset used in your controller. USB 2.0 requires the EHCI driver plus a USB 1.1 driver. See Table 5.

Table 5. Linux USB drivers

Table 5. Linux USB drivers	
Driver	Chipset
EHCI	USB 2.0 Support - requires one of UHCI, OHCI or JE
UHCI	Intel and VIA chipsets
JE	This is an alternate to UHCI for 2.4 kernels. If UHCI does not work, and you have an Intel or VIA chipset, try JE
OHCI	Compaq, most PowerMacs, iMacs, and PowerBooks, OPTi, SiS, ALi

We came across the `lsmod` command and the module configuration files /etc/modules.conf (2.4 kernel) and /etc/modprobe.conf (2.6 kernel in our earlier discussion of sound support. Listing 24 shows some of the modules associated with USB devices that are loaded on the same system as Listing 23. This system has a USB mouse

Listing 24. Using lsmod to show loaded USB modules

```
[root@lyrebird root]# lsmod | egrep 'usb|hci|hid|mouse|Module'
Module                  Size  Used by    Not tainted
usbserial              23420   0  (autoclean) (unused)
mousedev                5524   1
hid                    22244   0  (unused)
input                   5888   0  [keybdev mousedev hid]
ehci-hcd               20008   0  (unused)
usb-uhci               25740   0  (unused)
usbcore                77376   1  [usbserial hid ehci-hcd usb-uhci]
```

Note particularly that the usbcore module is used by all the other USB modules as well as the hid (human interface device) module.

Displaying USB information

So now we know something of the modules that support USB, how do we find out what USB devices are attached to our system? The information is to be found in the /proc/bus/usb part of the file system. The file /proc/bus/usb/devices contains summary information for currently attached USB devices. a partial listing for our system is shown in Listing 25.

Listing 25. Partial contents of /proc/bus/usb/devices

```
[root@lyrebird root]# cat /proc/bus/usb/devices
T:  Bus=04 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480 MxCh= 6
B:  Alloc=  0/800 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=01 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 2.04
S:  Manufacturer=Linux 2.4.21-32.0.1.EL ehci-hcd
S:  Product=Intel Corp. 82801DB USB2
S:  SerialNumber=00:1d.7
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   2 Ivl=256ms
T:  Bus=03 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
B:  Alloc=  0/900 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 0.00
S:  Product=USB UHCI Root Hub
S:  SerialNumber=1840
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=255ms
```

The *Spd=480* that we've highlighted above indicates a USB 2.0 bus while the *Spd=12* indicates a USB 1.1 (or possibly USB 1.0) device. Further down this listing our mouse is shown as having *Spd=1.5*. One and a half megabits per second should be fast enough for most mice.

As with other things that we have seen in the /proc file system, you will be pleased to know that there is a `lsusb` command to help you with display of this information. In particular, you can get a tree view of your USB devices by using the `-t` option. This shows their attachment hierarchy. You can use the `-d` option for information about a specific device if your system gives an abbreviated display using the `-t` option. The `-v` option produces verbose output which interprets many of the fields that we saw in Listing 25. For Listing 26, we've plugged in an external hub, a Nikon digital camera, a USB memory key and an external USB 2.00 hard drive and shown you some of the output.

Listing 26. Using the lsusb command

```
[root@lyrebird root]# lsusb -t
Bus#  4
`-Dev#   1 Vendor 0x0000 Product 0x0000
  |-Dev#   2 Vendor 0x0409 Product 0x0059
  | |-Dev#   8 Vendor 0x04b0 Product 0x0108
  | |-Dev#   4 Vendor 0x0d7d Product 0x1400
  | `-Dev#   7 Vendor 0x1058 Product 0x0401
  `-Dev#   3 Vendor 0x07d0 Product 0x1202
Bus#  3
`-Dev#   1 Vendor 0x0000 Product 0x0000
Bus#  2
`-Dev#   1 Vendor 0x0000 Product 0x0000
Bus#  1
```

```
`-Dev#   1 Vendor 0x0000 Product 0x0000
  `-Dev#   2 Vendor 0x1241 Product 0x1111
[root@lyrebird root]# lsusb -d 0x0409:0x0059
Bus 004 Device 002: ID 0409:0059 NEC Corp. HighSpeed Hub
[root@lyrebird root]# lsusb -d 0x04b0:0x0108
Bus 004 Device 008: ID 04b0:0108 Nikon Corp. Coolpix 2500
[root@lyrebird root]# lsusb -d 0x0d7d:0x1400
Bus 004 Device 004: ID 0d7d:1400 Phison Electronics Corp.
[root@lyrebird root]# lsusb -d 0x1058:0x0401
Bus 004 Device 007: ID 1058:0401 Western Digital Technologies, Inc.
[root@lyrebird root]# lsusb -d 0x07d0:0x1202
Bus 004 Device 003: ID 07d0:1202 Dazzle
[root@lyrebird root]# lsusb -d 0x1241:0x1111
Bus 001 Device 002: ID 1241:1111 Belkin Mouse
[root@lyrebird root]#
```

Listing 27 shows part of the verbose output available from the lsusb command. This is for a memory key. Note that the device has indicated its maximum power requirement (200mA). Note that this device will be treated as a SCSI device. Use either the `dmesg` command or the `fdisk -l` command to find out which SCSI device is mapped to a device. Most cameras equipped with USB ports, as well as card readers, flash devices and hard drives are treated as storage class devices and handled as SCSI devices in Linux. Many cameras come with Windows programs to help upload and pictures from the camera. In Linux you can simply mount the SCSI device representing the camera and copy the pictures to your hard drive where you can edit them with a program such as the GNU Image Manipulation Program (the GIMP). You can even erase files from the memory card or write files to it from Linux, allowing your camera to be used as an exotic replacement for a floppy disk.

Listing 27. Verbose output (partial) from lsusb command

```
[root@lyrebird root]# lsusb -vd 0x0d7d:0x1400

Bus 004 Device 004: ID 0d7d:1400 Phison Electronics Corp.
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x0d7d Phison Electronics Corp.
  idProduct          0x1400
  bcdDevice            0.02
  iManufacturer           1
  iProduct                2 USB DISK 12X
  iSerial                 3 0743112A0083
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           32
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0x80
    MaxPower              200mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           2
      bInterfaceClass         8 Mass Storage
```

```
      bInterfaceSubClass      6 SCSI
      bInterfaceProtocol     80 Bulk (Zip)
      iInterface              0
      ...
```

One more piece of information that is available to us now that we know the bus and device ids of your USB devices from Listing 26 is a way to determine which modules are required for a particular device. We'll illustrate a couple in Listing 28.

Listing 27. Verbose output (partial) from lsusb command

```
[root@lyrebird root]# usbmodules --device /proc/bus/usb/004/003
usb-storage
[root@lyrebird root]# usbmodules --device /proc/bus/usb/004/007
usb-storage
hid
```

Hot plugging

There are two commands that your system might use to handle hot plugging of USB devices, *usbmgr* and *hotplug*. According to which you are using, you will find configuration files in the /etc/usbmgr or /etc/hotplug directories. Newer systems are more likely to have hotplug.

Hot plugging for USB (and also PC cards) involves users plugging in devices while a system is running. The system then has to:

- Determine the device type and find a driver to run it
- Bind the driver to the device
- Notify other subsystems about the device. This allows disks to be mounted or print queues to be added for example.

Resources

Learn

- Review the entire LPI exam prep tutorial series on developerWorks to learn Linux fundamentals and prepare for system administrator certification.
- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- See the Reference Guide - Hard Disk Drives for a comprehensive history of hard drives. The Hard Disk Interfaces and Configuration section includes information on SCSI and a comparison of IDE/ATA and SCSI interfaces.
- The Linux documentation project is the home of lots of useful Linux documentation, including:
  - Large Disk HOWTO on disk geometry, the 1024 cylinder limit, and other limits for disks
  - Linux 2.4 SCSI subsystem HOWTO, covering SCSI on Linux, including device naming.
  - Linux SCSI Generic (sg) HOWTO on the new generic SCSI driver and utilities on Linux
  - The Network Administrators' Guide for networking on Linux
  - Linux Networking-HOWTO on SLIP, CSLIP, and PPP
  - Linux PPP HOWTO on setting up PPP on Linux
- Find more resources for Linux developers in thedeveloperWorks Linux zone.

Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Build your next development project on Linux with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

Ian Shields

Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. You can contact Ian at ishields@us.ibm.com.